Network Working Group
Request for Comments: 929

Joel Lilienkamp (SDC)
Richard Mandell (SDC)
Michael Padlipsky (Mitre Corp.)
December 1984

PROPOSED HOST-FRONT END PROTOCOL

Status Of This Memo

The reader should be aware of several things in regard to what the present document is up to. First and foremost, IT IS A PROPOSAL FOR A STANDARD, NOT A STANDARD ITSELF. Next, it assumes that the separate document, RFC 928, which is an introduction to the present document, has been read before it is. Next, it should be understood that "final cut" over this version of the document has been exercised by the author of RFC 928, not by the primary author of the present document, so any readers bothered by style considerations should feel free to blame the former, who's used to it, rather than the latter, who may well be guiltless. (Editing at a distance finally become too hard to manage, so if I'm typing it myself I'm going to fiddle with it myself too, including, but not limited to, sticking my own section on the Conceptual Model in before Joel's words start, rather than leaving it in the Introduction. MAP)

Finally, it should be noted that this is not a finished document. That is, the intent is eventually to supply appendices for all of the protocol offloadings, describing their uses of protocol idiosyncratic parameters and even their interpretations of the standard per-command parameters, but in order to get what we've got into circulation we haven't waited until all such appendices have been written up. (We do have notes on how to handle FTP, e.g., and UDP will be pretty straightforward, but getting them ready would have delayed things into still another calendar year, which would have been very annoying ... not to say embarrassing.) For that matter, it's not even a finished document with respect to what is here. Not only is it our stated intention to revise the protocol based upon implementation experience gained from volunteer test implementations, but it's also the case that it hasn't proven feasible to iron out all known wrinkles in what is being presented. For example, the response codes almost certainly need clarification and expansion, and at least one of us doesn't think mandatory initial parameters need control flags. However, to try too hard for polish would be to stay in subcommittee for the better part of forever, so what you see is what we've got, but certainly isn't meant to be what you or we are stuck with.

This RFC suggests a proposed protocol for the ARPA-Internet community, and requests discussion and suggestions for improvements. Distribution of this memo is unlimited.

Conceptual Model

There are two fundamental motivations for doing outboard processing. One is to conserve the Hosts' resources (CPU cycles and memory) in a resource sharing intercomputer network, by offloading as much of the required networking software from the Hosts to Outboard Processing Environments (or "Network Front-Ends") as possible. The other is to facilitate procurement of implementations of the various intercomputer networking protocols for the several types of Host in play in a typical heterogeneous intercomputer network, by employing common implementations in the OPE. A third motivation, of basing a network security approach on trusted mandatory OPEs, will not be dealt with here, but is at least worthy of mention.

Neither motivation should be allowed to detract from the underlying, assumed desire to perform true intercomputer networking, however. Therefore, it is further assumed that OPEs will be attached to Hosts via a flexible attachment strategy, as described in [1]. That is, at the software level an explicit Host-Front End Protocol (H-FP) will be employed between Hosts and OPEs, rather than having OPEs emulate devices or device controllers already "known" to Host operating systems (in order to avoid introducing new code into the Host).

For reasons discussed in the Introduction, an H-FP resolves into three layers. The Link layer enables the exchange of bits between Host and OPE. The Channel layer enables the bit streams to be demultiplexed and flow controlled (both the Channel and Link layers may use preexisting per-Host mechanizations, it should be recalled). The Command (or "Service Access") layer is our primary concern at present. It serves as the distributed processing mechanism which allows processes on Hosts to manipulate protocol interpreters (PIs) in OPEs on their behalf; for convenience, it will be referred to as "the H-FP" here. (It should be noted that the Link and Channel layers may be viewed as roughly equivalent to the inboard processing investment for a Host-comm subnet processor PI and device driver, so in practical terms the savings of resources achieved by outboard processing come from making the H-FP "smaller" than the inboard implementations of the protocols it allows to be offloaded.)

The crucial property of the H-FP conceptually is that it stands as the interface between a (Host) process and a PI (which is actually outboard). Usually, the model is that of a closed subroutine interface, although in some cases an interprocess communication mechanism model must be appealed to. That is, the interactions between cooperating H-FP PIs in some sense mimic subroutine or IPC calls, from the perspective of Host processes calling upon their own H-FP PIs, which in turn are of course interfacing via just such

mechanisms themselves. Another way of putting it is that "if the protocols were inboard," the processes invoking H-FP wouldn't know the difference. H-FP, then, may be viewed as a roundabout way of letting Host processes "get at" various PIs.

Naturally, the mechanization of the desired concept cannot be particularly literal. After all, the Hosts and the OPEs are different processors, so we're not envisioning a passing through of parameters in an exact fashion. However, in broad terms the model is just that of a somewhat funny interface between a process and a PI. (This should not be construed as ruling out the occurrence of events which prompt the OPE to initiate an exchange of commands with the Host, though; see the Introduction for more on the topic of "Symmetric Begins.")

Interaction Discipline

The interaction between the Host and the OPE must be capable of providing a suitable interface between processes (or protocol interpreters) in the Host and the off-loaded protocol interpreters in the OPE. This interaction must not, however, burden the Host more heavily than would have resulted from supporting the protocols inboard, lest the advantage of using an OPE be overridden.

Channel Level Interaction

As stated elsewhere, the Channel level protocol (implicitly in conjunction with the Link level) provides two major functions. These are demultiplexing the traffic from the Link level into distinct data streams, and providing flow control between the Host and the OPE on a per stream basis. These hold even if the Host-OPE attachment is DMA.

The data streams between the Host and the OPE are bidirectional. In this document, the basic unit of data transferred by the Channel level is referred to as a "chunk". The primary motivation for this terminology is that the H-FP permits the Channel level to be one of several possible protocols, each with its own terminology. For example, a chunk on an X.25 Channel would be a packet, while a chunk on the DTI H-FP channel would be a message. While the Command level is, in a sense, "more efficient" when the chunk size is permitted to be large, the flexibility permitted in the choice of protocols at the Channel level precludes any assumptions about the chunk size.

Each data stream is fully asynchronous. A Channel protocol user can send data at any time, once the channel has been properly opened. (The Command level's logic may render some actions meaningless, however.) The data transfer service provided by the Channel protocol

is reliable; this entails delivery in the correct order, without duplication, and checked for bit errors. All retransmission, error checking, and duplicate detection is provided by this protocol in a way that is transparent to the user. (If the attachment is DMA, stream identification and chunk length must still be provided for.)

The flow control at the Channel level is provided to prevent the OPE and the Host from overloading each other's resources by excessive transmissions. In general, this flow control should not directly affect the outboard protocol interpreters' operation. On the other had, this flow control has the same effect as explicit interface events that provide flow control between the user and the protocol interpreter (e.g., the Allocate event of the interface specification for TCP found in MIL-STD 1778). Hence, such events do not need to be communicated explicitly at the Command level. (If the attachment is DMA, flow control must still be provided for.)

Should Hosts require an OPE to be attached via a Link Level that furnishes physical demultiplexing (e.g., a group of RS232 ports), any attempt to avoid furnishing reliability and explicit flow control, is done at their peril; we have not chosen to assist such an enterprise, but neither have we precluded it. (It would certainly violate the spirit of the thing, however.)

Command Level Interaction

The approach chosen for this H-FP is to base the interaction on a small set of commands, separately applicable to a given Channel Level channel. The commands are simple, but sufficiently flexible to permit the off-loading of the interpreters of the large number of protocols at various levels in the hierarchy. This flexibility is made possible in part by the similar nature of the interfaces to most protocols, combined with the provision of "protocol idiosyncratic parameters". These parameters are defined for each offloaded protocol interpreter in the OPE. The use of such parameters does not complicate the basic design of the OPE, since it must be customized for each off-loaded protocol anyway, and all that is required of the OPE for those parameters is to pass them to the off-loaded protocol interpreter. Hence, an interface tailored to a particular protocol can be created in a straightforward and cost-effective way.

The command dialog is more or less asynchronous. Commands can be issued at any particular time (except when there is a pending command, which will be discussed below), and there is no need for dummy traffic on a channel when no commands are issued.

Associated with each command is a response. The purpose of this

response is to indicate, at some level that depends in part on the particular protocol interpreter that is offloaded to the OPE, whether the command was successfully executed, and if unsuccessful, the reason. Often, generating the response involves interaction with the protocol interpreter before a response can be generated.

When a command is issued, the issuer must wait for a response before another command is issued. The nature of the communication between the Host and the OPE is thus a lock step command/response dialog. There are two major exceptions to this principle, however. One exception is the abrupt form of the End command, which can be issued at any time to cancel any previously issued commands, and indicate that services are no longer desired. The other exception is the Signal command. Since a Signal is out-of-band and usually of high importance, forcing it to wait on a response would be undesirable. Hence, a Signal command can be issued while commands (other than Signal) are pending. However, a Signal command should not be issued before a successful response to the Begin command has been received. Since it is possible for more than one command of different types to be pending at one time, a mechanism to distinguish responses is needed. Since there are never two commands of the same type pending, including the command name in the response is sufficient to make this distinction.

A special case command is the Transmit command. Details of the Transmit command are provided in the next section. Essentially, the Transmit command is used to invoke the data transfer services of the off-loaded protocol (when issued by the Host) or to indicate the arrival of new data from the network (when issued by the OPE). The nature of specific protocol interfaces for these events varies widely between protocols. Some may block until the data is accepted by the remote counterpart (or "peer") protocol interpreter, while others may not. Hence, there is a special parameter which indicates the nature of the Transmit command interface. It can either require that the response should be generated immediately after determining the Transmit command is complete and formed properly, or can indicate that the response should not be generated until the appropriate interface event is given by the remote protocol interpreter. The default action for all Transmit commands can be initialized using the Begin command and changed using the Condition command. Also, the default action can be temporarily overridden by specifying a parameter with the Transmit command. The net result of this mechanism is to allow the Host to determine within reason just how lock-stepped transmissions are to be. (It is assumed that the usual case will be to transfer the burden of buffering to the OPE by taking immediate responses, provided that doing so "makes sense" with the particular offloaded protocol in play.)

Some protocols provide a block-oriented data transfer service rather than a stream-oriented one. With such a service, the data associated with a transfer request is viewed as an integral unit. For actual network transmission, the protocol may permit these units to be grouped or fragmented. However, the receiving end must deliver the data in the original, integral units. Protocols that conform to this model include some datagram protocols such as IP and UDP, and also some connection protocols such as NBS TP.

To cater to these types of protocols, it is a convention that commands, their parameters, and any associated data be transferred between the Host and the OPE in a single chunk. Any data associated with an H-FP command is viewed as an integral unit which is used in the corresponding service request given to the outboard protocol interpreter or delivered as a complete unit to the process in the Host. Operation of stream-oriented protocols such as TCP will not be adversely affected by this convention.

To accommodate Channel protocols that do not provide for arbitrarily large chunks, a mechanism at the Command level is required to permit the linking of multiple chunks into a single command, in order to transfer the burden of buffering as much as possible from the Host to the OPE. The facility proposed here would consist of an indication at the beginning of each chunk which would distinguish integral commands, fragments of a command for which more fragments are yet to arrive, and the final fragment of a command. The details of this mechanism are discussed in the section on the syntax of commands and responses.

It is a convention for this H-FP that any data associated with a command must start on a word boundary (as defined by the local system). Consequently, there is a need to provide padding within the commands. Such padding is used only to fill to the next appropriate boundary, and has no semantic significance to the command interpreter (i.e., two commands that are identical except for the amount of padding should behave identically). The details of this padding are discussed in the section on the syntax of commands and responses.

Syntax Rules

At the Command Level, communication between the Host and the OPE takes the form of commands and responses. A command is a request for some particular action, and the response indicates the success or failure of performing the requested action.

All commands and responses are coded in ASCII characters. (Nothing precludes OPEs from accepting EBCDIC from Hosts that use it in native mode, but that is not required.) These characters are sent in some way convenient for the Host, and the OPE is sufficiently flexible to interpret them. (i.e., OPEs are expected to accommodate Host idiosyncracies in regard to such things as use of 7-bit ASCII in a 9-bit field.) This approach offers several advantages:

Adaptabilities in most Hosts: Most Hosts have the ability to generate and interpret ASCII character streams. Hence, integrating H-FP into a Host will not require difficult software.

Script generation: Generation of test and operational command scripts will be simplified, since they will not need to contain special characters.

Terminal Operation: Using simple command streams simplifies the conversion of an OPE to a generic virtual terminal support machine. This is particularly useful during development and testing.

Testing: Testing will not require special hardware to interpret commands and responses. A terminal or data line analyzer would be adequate.

The specific format for the commands and responses will be discussed in the sections that follow. In those sections, the quote character is used to indicate strings. The symbols "<" and ">" (referred to as angle brackets) are used as meta-characters.

Syntax of Commands

As alluded to in the section discussing the interaction discipline between the Host and the OPE, a function is provided by which a chunk can be used to carry either a complete command or a fragment of a command. The mechanism chosen to provide this function entails use of the first character position in the chunk as a chunk usage identifier. The character "C" in the first position indicates a chunk containing a single, complete command. "F" in the first position indicates a chunk which is the first part of a multichunk command. "M" in the first position indicates the chunk is a middle

part (neither the first nor the last chunk) of a command. Finally, "L" indicates the chunk is the last chunk of a multi-chunk command. Hence, the following sequences of chunks (the letter corresponds to the chunk usage identifier in each chunk, and the angle brackets enclose a chunk) are legal:

<C>
<F><L>
<F><M><M><L>

while the following are not legal:

<L>
<M><L>
<F><C>

Tactics for handling multiple chunks with regard to OPE buffering limits are left to the ingenuity of OPE builders. The spirit is to take as much as you can, in order to relieve the Host of the necessity of buffering itself.

A command always begins immediately following the indicator character, with possible intervening spaces. This implies a chunk can contain at most one complete command. The end of the command (not including the data) is signified by a newline (denoted as <nl> in this document) that does not appear inside a quoted string (see below). The end of the data is designated by the end of the last chunk.

Commands take the form of an ASCII string. The command identifier is the first word of the chunk. It consists of at least the first two letters of the command, in either upper or lower case (e.g., the sequences "BE", "Be", "bE", and "be" all identify the Begin command). Additional letters of the command name can be included if desired to aid readability of the command stream.

Following the command identifier is a list of parameters. These parameters are also represented as ASCII strings, although the specific format will depend on the particular parameter. The data to be transmitted is not considered a control parameter, however, and need not be ASCII data.

Parameters are separated by one or more spaces. Tabs, newlines, and other white space are not legal parameter separators.

Parameter strings may be quoted, using the character <">. Any

characters between the <"> characters are a part of the parameter, including spaces and newlines. The character <"> that is part of the parameter is represented inside a quoted string as <"">.

The order in which the parameters appear within the command is significant to their interpretation by the Host and by the OPE. Optional parameters may be skipped by using the characters ",," to indicate a NULL parameter. Such a NULL parameter takes its default value. Alternatively, each parameter has a MULTICS/UNIX style Control Argument/Flag associated with it that can be used to identify the parameter, without placing NULL parameters for each parameter skipped. This flag consists of one or two ASCII characters, and either upper or lower case may be used. For example, if the fourth parameter of a command had a flag of "-p" and the user wished the first three parameters to be null, he could use:

```
command -p value
```

or

```
command -P value
```

instead of

```
command ,, ,, ,, value
```

if it were more convenient for the Host to do so. Flagged parameters must still appear in the correct sequence within the command, however.

There may be data associated with some of the commands. Any such data is placed into the chunk following all the parameters and the unquoted newline. Padding can be provided by placing spaces between the end of the final parameter string and the newline, so that data begins on a word boundary. The OPE will always pad to a host word boundary. Padding by hosts is optional.

Syntax of Responses

Responses are actually just a special form of a command. It is anticipated that all responses would fit into a single channel chunk, although the mechanisms described for multichunk commands can certainly be used in responses. The ASCII string used to uniquely identify the response command is "RE" ("Re", "rE", and "re" are also permitted).

After the response command identifier is the original command

identifier, so the response can be associated with the proper command. Following this identifier is a three ASCII digit response code, a set of protocol idiosyncratic parameters, and a textual message. The protocol idiosyncratic parameters are used to transfer interface information between the Host and the OPE, and may not be needed when off-loading some protocol interpreters. The textual message is intended for human interpretation of the response codes, and is not required by the protocol. The three digits uniquely identify the semantics of the response, at least within the context of a particular command and particular outboarded protocol interpreter.

Responses are numerically grouped by the type of information they convey. The first digit identifies this group, and the last two digits further qualify the reply. The following list illustrates this grouping.

0XX Successful: The command was executed successfully. The response code may contain further information.

1XX Conditional Success: The command was executed successfully, but not exactly according to the service and flow control suggestions. If those suggestions were particularly important to the requester, he may wish to issue an End command. The response code contains information on what suggestion or suggestions could not be followed.

2XX Command Level Error: An error at the command level has occurred. This could include requesting services of a protocol not supported, or a problem in the way those services were requested. This level does not include problems with the syntax of the command or its parameters.

3XX Syntax and Parameter Errors: An error in the syntax of the command or a problem with one of its parameters has occurred. A problem with a parameter may be other than syntactical, such as illegal address.

4XX Off-loaded Protocol Interpreter Problems: Some problem with the particular off-loaded protocol has occurred.

5XX Local OPE Internal Problems: Problems, such as insufficient OPE resources, or problems with OPE to subnet interface.

6XX Security Problem: Some problem with Host, network, or OPE security has occurred. The response code indicates the problem.

7XX Reserved for Future Expansion

8XX Reserved for Future Expansion

9XX Protocol Idiosyncratic Errors: Some error occurred that is idiosyncratic to the particular off-loaded protocol being used. The response code indicates the error.

Description of the Commands

As stated above, communication between the Host and the OPE at the Command Level is accomplished using commands and responses. Commands may be issued by either the Host or the OPE, and are used to stimulate activity in the other entity. Some commands may only have a meaningful interpretation in one direction, however. A response indicates that the activity started by the command was completed, and a code indicates success or failure of the command, and perhaps other information related to the command as well.

Associated with each command is a set of parameters. The order in which the parameters appear is significant to the correct operation of the protocols. More information on the syntax of command parameters can be found in the syntax descriptions.

The commands are:

- Begin: initiate communication between a process in the Host and an off-loaded protocol interpreter in the OPE. (A Channel level stream/connection will typically have been opened as a prior step. All other commands, except No-op, apply to a stream on which a successful Begin has been done.)

- Transmit: transmit data between a process in the Host and an off-loaded protocol interpreter in the OPE.

- Signal: cause an out-of-band signal to be sent by the off-loaded protocol interpreter to its peer, or indicate the arrival of such a signal from the remote side.

- Condition: alter the off-loaded protocol interpreter's operational characteristics.

- Status: transfer status requests or information between a process in the Host and an off-loaded protocol interpreter in the OPE.

- End: indicate that services from the off-loaded protocol interpreter are no longer required, or will no longer be provided.

- No-op: performs no operation, but facilitates testing.

These commands will be discussed in the following sections. Each of these sections includes a discussion of the purpose of the command, a description of each of the parameters used with the command, a list of responses for the command, an example of the command, and a set of notes for the implementor. (An appendix will eventually be furnished for each protocol offloading, showing the use of its protocol idiosyncratic parameters as well as of the general parameters on a per-command basis. Initially, only representative offloadings will be treated in appendices, with others to be added after the protocol gains acceptance.)

Begin

Purpose of the Begin Command

The purpose of a Begin command is to initiate communication between the Host and the OPE on a particular stream or channel (the channel is opened as a separate step, of course). The interpretation of the command is somewhat dependent upon whether it was issued by the Host of the OPE.

- If the command was issued by the Host, it means some process in the Host is requesting services of a protocol that was off-loaded to the OPE. The user request results in the establishment of a channel connection between the Host and the OPE, and a Begin command to the Command interpreter in the OPE.

- If the command was issued by the OPE, it means some protocol interpreter in the OPE has data for some process in the Host which is not currently known by the OPE. An example would be an incoming UDP datagram on a new port, or if no Begin for UDP had been issued at all by the Host. (An incoming TCP connection request could be handled by a response to the user's Passive Open request, which had previously caused a Begin request from the Host; an incoming TCP connection request to a port on which no Listen had been issued would cause an OPE generated Begin, however.)

As indicated earlier, any particular Host is not required to support two-way Begins.

Parameters of the Begin Command

The Begin command has several parameters associated with it. These parameters contain information needed by the offloaded protocol to provide an adequate level of network service. This information includes protocol, source and destination addresses, and also type of service and flow control advice. These parameters are discussed in detail below.

Protocol

The protocol parameter identifies that off-loaded protocol in the OPE to which Begin is directed, or which issued the Begin to the Host. For example, if the user wished to utilize TCP services, and the TCP software was off-loaded into the OPE, then the Protocol parameter for the Begin command would be TCP.

There are two categories of protocol parameters -- generic and specific. A generic parameter identifies a type of protocol service required, but does not identify the actual protocol. Use of generic protocols allows a Host process to obtain network services without specific knowledge of what protocol is being used; this could be appropriate for use in situations where no specific aspect(s) of a specific protocol is/are required. For example, the user may select a generic Host-to-Host connection protocol, and (at some point in the future) may actually receive services from either TCP or the NBS Transport Protocol, depending on the network (or even the foreign Host) in question. A specific protocol parameter identifies some particular protocol, e.g., TCP, whose use is required for the given channel.

The valid entries for the protocol field include:

Generic	Specific	Comment
GIP	IP	Datagram Internetwork Protocol
HHP	TCP	Connection Transport/Host-Host Protocol
GDP	UDP	Datagram Transport/Host-Host Protocol
VTP	TEL	Virtual Terminal (Telnet) Protocol
GFP	FTP	File Transfer Protocol
MAIL	SMTP	Mail Transfer Protocol
PROX	PROX	Proximate Net Interface Protocol

(Note that the final line is meant to allow for a process in an OPE'd Host's getting at the PI of the Network Interface Protocol for whatever the proximate network is. Of course, so

doing only makes sense in specialized contexts. We conceive of the desirability of "pumping bits at a peripheral" on a LAN, though, and don't want to preclude it, even if it would be impossible on many LAN's to deal with the problem of distinguishing traffic coming back on the LAN in this "raw" mode from normal, IP traffic. Indeed, in some contexts it is likely that administrative considerations would preclude avoidance of IP even if technical considerations allowed it, but it's still the case that "the protocol" should provide a hook for going directly to the L I protocol in play.)

There is no default value for this parameter. If it is not present, the Begin command is in error. The control flag for this parameter is -pr.

Active/Passive

The Active/Passive parameter indicates whether the issuer of the Begin command desires to be the Active or Passive user of the protocol. This parameter is particularly relevant to connection-oriented protocols such as TCP, where the user may actively pursue connection establishment, or else may passively wait for the remote entity to actively establish the connection; it also allows some process to establish itself as the Host "fielder" of incoming traffic for a connectionless protocol such as IP.

Active is requested using the single character "A". Passive is indicated using the character "P". The default value of this parameter is "A". Also, when the OPE issues the Begin command, the value must be "A". The control flag for this parameter is -ap.

Foreign Address Primary Component

The addressing structure supported by H-FP is two level. Each address has two components, the primary and the secondary. The exact interpretation of these two components is protocol specific, but some generalities do apply. The primary component of the address identifies where the protocol is to deliver the information. The secondary component identifies which recipient at that location is to receive the information. For example, the TCP primary address component is the Host's Internet Address, while the secondary address component is the TCP port. Similarly, IP's primary address component is the Host's Internet Address, and the secondary address component is the IP ULP field. Some protocols provide only a single level

of addressing, or the secondary level can be deduced from some other information (e.g., Telnet). In these cases, only the primary component is used. To cater to such cases, the secondary component parameter comes later in the parameter list.

The Foreign Address Primary Component parameter contains the primary component of the destination address. It may be in either a numeric or symbolic form. (Note that this allows for the OPE to exercise a Name Server type of protocol if appropriate, as well as freeing the Host from the necessity of maintaining an in-board name to address table.) The default value for this parameter, although it only makes sense for Passive Begins, is "Any Host". The control flag for this parameter is -fp.

Mediation Level

The mediation level parameter is an indication of the role the Host wishes the OPE to play in the operation of the protocol. The extreme ranges of this mediation would be the case where the Host wished to remain completely uninvolved, and the case where the Host wished to make every possible decision. The specific interpretation of this parameter is dependent upon the particular off-loaded protocol.

The concept of mediation level can best be clarified by means of example. A full inboard implementation of the Telnet protocol places several responsibilities on the Host. These responsibilities include negotiation and provision of protocol options, translation between local and network character codes and formats, and monitoring the well-known socket for incoming connection requests. The mediation level indicates whether these responsibilities are assigned to the Host or to the OPE when the Telnet implementation is outboard. If no OPE mediation is selected, the Host is involved with all negotiation of the Telnet options, and all format conversions. With full OPE mediation, all option negotiation and all format conversions are performed by the OPE. An intermediate level of mediation might have ordinary option negotiation, format conversion, and socket monitoring done in the OPE, while options not known to the OPE are handled by the Host.

The parameter is represented with a single ASCII digit. The value 9 represents full OPE mediation, and the value 0 represents no OPE mediation. Other values may be defined for

some protocols (e.g., the intermediate mediation level discussed above for Telnet). The default value for this parameter is 9. The control flag for this parameter is -m.

Transmit Response Discipline

The Transmit Response Discipline parameter is used to set the desired action on the OPE's part for generating responses to Transmit commands. Essentially the parameter determines when the OPE's response to the transmit command occurs (i.e., immediately or delayed).

The Transmit Response Discipline value is represented by a single ASCII character. The character "N" is used for nonblocking Transmit commands, which implies that responses for Transmit commands should be generated as soon as the command has been examined for correctness (i.e., that the syntax is good and the parameters appear reasonable). In other words, the outboard protocol interpreter has the data in its queue, but hasn't necessarily transmitted it to the net. The character "B" is used for blocking Transmit commands, which requests that the response not be generated until the protocol interpreter has successfully transmitted the data (unless, of course, the Transmit command was badly formed). The default value for this parameter is "N", or a nonblocking Transmit command. The control flag for this parameter is -tr. (Depending on the protocol in play, "successfully transmitted" might well imply that an acknowledgment of some sort has been received from the foreign Host, but for other protocols it might only mean that the given collection of bits has been passed from the OPE to the proximate net.)

Foreign Address Secondary Component

The addressing mechanisms supported by this level of H-FP are discussed above. The Foreign Address Secondary Component parameter contains the value of the destination address's secondary component. Some protocols do not require this parameter, or can obtain it from other information. Therefore, the default value for this parameter is NULL. A NULL secondary component might be an error for some protocols, however. The secondary component can be expressed either numerically or symbolically. The control flag for this parameter is -fs. (Note that it is intended to be "legal" to specify a Secondary Component other than the Well-Known Socket for the protocol in play; in such cases, the result should be that the virtualizing of the given protocol be applied to the stream, in the

expectation that that's what the other side is expecting. This is to cater to, for example, a Terminal-Terminal protocol that merely "does Telnet" to a socket other than the usual Logger.)

Local Address Secondary Component

The Local Address Secondary Component parameter contains the value of the local address's secondary component. (The primary component is assumed to be the default for the Host, but can be altered as well; see below.) Some protocols do not require this parameter, or can obtain it from other information. In some cases, the OPE may already know the value for this parameter and therefore not require it. The default value of this parameter is NULL. The local address secondary component can be expressed either numerically or symbolically. The control flag for this parameter is -ls.

Begin Timeout Interval

After a Begin command is issued, a timer can be started. If the activity requested cannot be performed within some timed interval, then the Begin command may expire. An expired Begin command returns a response code indicating a Begin timeout occurred. The Begin Timeout Interval parameter contains the length of time the timer will run before the Begin timeout occurs.

The parameter is represented as a string of ASCII digits indicating the time interval in seconds. The default value of this parameter is infinity (i.e., the Begin command will never timeout). The control flag for this parameter is -bt.

Type of Service Advice

The Type of Service Advice parameter contains information on the service characteristics the user desires from the offloaded protocol. Included in this parameter is the precedence of the data transfer, and also indication of whether high throughput, fast response time, or low error rate is the primary goal.

The format of this parameter is a letter immediately (i.e. no intervening spaces) followed by a digit. The letter "T" indicates that high throughput is desired. The letter "R" indicates minimal response time is the goal. The letter "E" indicates that low error rates are the goal. The letter "N" indicates there are no special service requirements to be conveyed. The digit immediately following the character

indicates the desired precedence level, with zero being the lowest, and nine being the highest. The specific interpretation of this parameter is dependent on what service options are provided by the protocol. The default value of this parameter is the lowest precedence (ROUTINE), and no special service requests. The control flag for this parameter is -ts.

Flow Control Advice

The Flow Control Advice parameter contains information on the flow characteristics desired by the user. Some applications such as file transfer operate more efficiently if the data is transferred in large pieces, while other, more interactive applications are more efficiently served if smaller pieces are used. This parameter then indicates whether large or small data blocks should be used. It is only relevant in stream or connection-oriented protocols, where the user sends more than a single piece of data.

This parameter is represented by a single ASCII digit. A value 0 means the data should be sent in relatively small blocks (e.g., character or line oriented applications), while a value 9 means the data should be sent in relatively large blocks (e.g., block or file oriented applications). Other values represent sizes between those extremes. The character "N" indicates that no special flow control advice is provided. The actual interpretation of this parameter is dependent on the particular protocol in the OPE. The default value of this parameter is no flow control advice. In this case, the protocol in the OPE will operate based only on information available in the OPE. The control flag for this parameter is -fc.

Local Address Primary Component

This parameter contains the local address primary component. It is anticipated that under most circumstances, this component is known to both the Host and the OPE. Consequently, this parameter is seldom required. It would be useful if the Host desired to select one of several valid addresses, however. The control flag for this parameter is -lp.

Security

The security parameters contain a set of security level, compartment, community of interest, and handling restriction information. Currently, security is provided by performing all

processing at system high level or at a single level. Consequently, these parameters are probably redundant, since the security information is known. In the future, however, these parameters may be required. Therefore a field is provided. The control flag for this parameter is -s.

Protcol Idiosyncratic Parameters

The remaining parameters are protocol idiosyncratic. That is, each protocol that is off-loaded may have a set of these parameters, which are documented with a description of the off-loaded protocol. The default value for these parameters is NULL, unless otherwise specified by a particular offloaded protocol. The control flag for this set of parameters is -pi, which identifies the first protocol idiosyncratic parameters. Control flags for other protocol idiosyncratic parameters must be defined for each off-loaded protocol.

Data

After the Protocol Idiosyncratic Parameters, if any, and the required <nl>, if the protocol in play allows for it at this juncture the rest of the chunk will be interpreted as data to be transmitted. That is, in connection oriented protocols data may or may not be permitted at connection initiation time, but in connectionless protocols it certainly makes sense to allow the H-FP Begin command to convey data. (This will also be useful when we get to the Condition command.)

Responses

The following responses have been identified for the Begin command:

000	Command completed successfully
101	Throughput not available; using maximum
102	Reliability not available; using maximum
103	Delay not available; using minimum
110	Flow Control advice not followed; smaller blocks used
111	Flow Control advice not followed; larger blocks used
201	Failed; Begin not implemented in this direction
202	Failed; timeout
203	Failed; Begin command on already active channel
300	Problem with multiple chunks
301	Syntax problem with Begin command
302	Protocol not supported in OPE/Host
303	Active service not available

304	Passive service not available
305	Invalid Foreign Address Primary Component
306	Invalid Transmit Discipline
307	Invalid Foreign Address Secondary Component
308	Invalid Local Address Secondary Component
309	Invalid Timeout Interval
310	Invalid Type of Service Advice
311	Invalid Flow control Advice
312	Invalid Local Address Primary Component
401	Protocol Interpreter in OPE not responding
402	Remote Protocol Interpreter not available
403	Failed; insufficient protocol interpreter resources
501	Failed; insufficient OPE resources
601	Request violates security policy
602	Security parameter problem

Additionally, protocol idiosyncratic responses will be defined for each off-loaded protocol.

Example of Begin Command

The Begin command is the most complex of the H-FP Command Level. When the off-loaded protocol is TCP, the Begin command is used to open TCP connections. One possible example of a Begin command issued by an inboard Telnet interpreter to open a TCP connection to ISIA, with no begin timeout interval, is:

```
C BE TCP A ISIA 9 N 23 ,, ,, NO S <nl>
```

Where:

TCP	The code for the protocol TCP
A	Indicates Active Begin
ISIA	The name of a Host at USC-ISI
9	Mediation Level 9: Full OPE mediation
N	Non-blocking transmit
23	Destination Telnet Port
,,	skip over parameters (Local Address Secondary, Begin Timeout Interval)
NO	Type of Service Advice: No special Advice, Normal Precedence
S	Flow Control Advice: use small blocks

This command will cause the OPE to invoke the TCP interpreter to generate the initial SYN packet to the well-known Telnet socket on Host ISIA. It also informs the OPE to do all TCP related processing via the Mediation Level, accepts default

Local Address parameters, and sets the Begin Timeout Interval to infinity. The precedence of the TCP connection is Normal, and the TCP interpreter is informed that the data stream will consist of primarily small blocks.

Notes to the Implementor

Response 203 might seem silly to some readers, but it's there in case somebody goofed in using the Channel Layer.

Transmit

Purpose of the Transmit Command

The purpose of the Transmit command is to permit the process in the Host to send data using an off-loaded protocol interpreter in the OPE, and also to permit the OPE to deliver data received from the network destined for the process in the Host. The Transmit command is particularly relevant to connection and stream type protocols, although it has applications for connectionless protocols as well. After the Begin command is issued successfully and the proper Response received, Transmit commands can be issued on the given channel. The semantics of the Transmit command depend on whether it was issued by the Host or the OPE.

- If the Host issues the Transmit command, a process in the Host wishes to send the data to the destination specified to the off-loaded protocol interpreter that was established (typically) by a previous Begin command on the given H-FP channel.

- If the OPE issues the command, the OPE has received data destined for a process in the Host from a connection or stream supported by the off-loaded protocol that was established by a previous Begin command on the given H-FP channel.

Parameters of the Transmit Command

The Transmit command has one parameter associated with it. It is an optional parameter, to temporarily override the response discipline for this particular transmit command. Some protocols may have protocol-idiosyncratic parameters as well. The transmit command also has data associated with it. All parameters must precede the data to be transmitted.

Response Discipline Override

The Response Discipline Override parameter indicates the desired response discipline for that individual Transmit Command, overriding the default response discipline. A single ASCII character is used to indicate the desired discipline. The character "N" indicates that this Transmit command should not block, and should return a response as soon as the data is given to the protocol interpreter in the OPE. The character "B" indicates that this Transmit command should block, meaning that a response should not be generated until the data has been sent to the destination. The default value of this parameter is the currently defined Transmit Command response discipline. The use of this parameter does not alter the currently defined Transmit command response discipline; the default is changed with the Condition command. The control flag for this parameter is -rd.

Protocol-Idiosyncratic Parameters

Any other parameters to the Transmit command are protocol-idiosyncratic. That is, each protocol that is off-loaded has a set of these parameters, which are documented with a description of the off-loaded protocol. The default value for these parameters is NULL, unless otherwise specified by a particular off-loaded protocol. The control flag for this set of parameters is -pi, which identifies the first protocol-idiosyncratic parameters. Control flags for other protocol-idiosyncratic parameters must be defined for each off-loaded protocol.

Responses

The following responses for the Transmit command have been identified:

000	Transmit Command completed successfully
201	Transmit Command not appropriate
300	Problem with multiple chunks
301	Syntax problem with Transmit Command
302	Invalid Transmit Command Response Discipline
401	Protocol Interpreter in OPE not responding
402	Failure in remote protocol interpreter
403	Failed; insufficient protocol interpreter resources
501	Failed; insufficient OPE resources
601	Request violates security policy

Additionally, protocol-idiosyncratic responses will be defined for each off-loaded protocol.

Example of Transmit Command

The transmit command is used in TCP to provide the TCP write call. An example of such a transmit command would be:

C TR N <nl> <DATA>

Where N indicates non-blocking transmission discipline, <nl> is the required command-ending newline, and <DATA> is presumed to be the user's data that is to be transmitted.

Notes to the Implementor

If you get a 403 or a 501 response and have sent a multiple chunk it probably makes sense to try a single chunk; if you've sent a single chunk, it makes sense to wait a while and try again a few times before giving up on the stream/channel.

Condition

Purpose of the Condition Command

The primary purpose of the Condition command is to permit a process to alter the characteristics that were originally set up with the Begin command. (That is, "condition" is a verb.) These characteristics include the addresses, the mediation level, the type of service, and the flow control parameters from Begin. They may also include protocol-idiosyncratic characteristics. (Although Condition is usually thought of as a Host->OPE command, it may also be used OPE->Host in some contexts.)

Condition is a generic command that may find little use in some off-loaded protocols. In others, only some of the parameters identified may make sense. For example, changing the destination address of a TCP connection involves closing one connection and opening another. Consequently, in may make more sense to first issue an End command, and then a Begin with the new address. In other protocols, such as IP or UDP, changing the address on each datagram would be a perfectly reasonable thing to do.

Parameters of the Condition Command

The Condition command has the same parameters as the Begin command. Any parameters expressed in a Condition command indicate the new values of the characteristics to be altered; all parameters not expressed retain the current value.

Although it is possible to express the change of any of the characteristics originally set up in the Begin command using the Condition command, there are some characteristics that do not make sense to alter, at least for some protocols. For example, once a connection is opened, it does not make much sense to change the Foreign Address Primary or Secondary Components. Doing so is inconsistent with current versions of TCP, and would require the closing of the existing connection and opening a new one to another address. Earlier versions of TCP did permit connections to be moved. If a protocol that provided such a feature was implemented in the OPE, the changing the Secondary Address Components would be a reasonable thing to do.

Responses

The responses to the Condition command are the same as those to the Begin command.

Example of Condition Command

The Condition Command can be quite complex, and can be used for many purposes. One conceived use of the condition command would be to change the type of service advice associated with the channel. An example of this (which also demonstrates the ability to skip parameters) is:

```
C -ts T <nl>
```

which causes the offloaded PI associated with the current channel to attempt to achieve high throughput (in its use of the comm subnet(s) in play).

Notes to the Implementor

Signal

Purpose of Signal Command

The purpose of the Signal Command (implicitly at least) is to permit the transfer of out-of-band signals or information between the Host and the OPE, in order to utilize (explicitly) out-of-band signaling services of the off-loaded protocol. The semantics of the Signal command depend upon whether it was issued by the Host or the OPE.

- If the Signal command was issued by the Host, it means a process in the Host desires to send out-of-band data or an out-of-band signal.

- If the Signal command was issued by the OPE, it means out-of-band data or an out-of-band signal arrived for the process associated with the channel in the Host.

Parameters of the Signal Command

The basic usage of the Signal command is with no parameters, which sends or reports the receipt of an out-of-band signal. Some protocols, such as the NBS Transport Protocol, permit the user to send data with the out-of-band signal. Hence, data is permitted to accompany the Signal command. There may also be protocol-idiosyncratic parameters for the Signal command. If this is the case, these parameters would come before the data.

Protocol-Idiosyncratic Parameters

The parameters for the Signal command are protocol idiosyncratic. That is, each protocol off-loaded has a set of these parameters. The default value for these parameters is their previous values. Control flags for multiple protocol-idiosyncratic parameters must be defined for each off-loaded protocol.

Responses

The following responses have been identified for the Signal command:

000	Command completed successfully
201	Command not appropriate
300	Problem with multiple chunks
301	Syntax problem with Command

401 Protocol Interpreter in OPE not responding
402 Failure in remote protocol interpreter
403 Failed; insufficient protocol interpreter resources
501 Failed; insufficient OPE resources
601 Request violates security policy

Additionally, protocol-idiosyncratic responses will be defined for each off-loaded protocol.

Example of Signal Command

The major perceived use for the Signal command when offloading a connection protocol is sending an out-of-band signal with no data. In such a case, the appropriate signal command would be:

C SI <nl>

Notes to the Implementor

Some protocols may allow only only one outstanding signal at a time. For these protocols, it is an implementation issue whether the OPE will buffer several signals, but a good case could be made for the position that a scrupulous OPE would reflect a 202 response back to the Host in such cases.

There is some question as to the proper handling of the "expedited data" notion of some (particularly ISO) protocols. It might be more appropriate to deal with such a thing as a protocol idiosyncratic parameter on the Transmit command instead of using the Signal command (even if it's the closest approximation to an out-of-band signal in the given protocol). If it's provided using the Signal command, the expedited data should not be passed as ASCII, and should appear after the command-terminating newline character (and appropriate padding with space characters).

Status

Purpose of Status Command

The purpose of the Status command is to permit the Host to request and obtain status information from the OPE, and vice versa. This includes status request of a conventional protocol interface (e.g., in TCP, there is a request to determine the state of a particular connection).

Parameters of the Status Command

The parameters for the Status command indicate whether it is a request or a response, and contain the status information.

Request/Report

This parameter indicates whether the command is a Status request or a Status report. It consists of a single ASCII character. Q indicates a request (query), and R indicates a report. It should be noted that a report may be generated as the result of a query, or may be generated as the result of specific protocol mechanisms.

Protocol-Idiosyncratic Parameters

The parameters to the status command are protocol-idiosyncratic. That is, each protocol off-loaded has a set of these parameters. The default value for these parameters is their previous values. Among these parameters is an identifier of the type of status information contained or requested, and a value or set of values that contain the particular status information. The status information itself should be the last item in the command. The control flag for this set of parameters is -pi, which identifies the first protocol-idiosyncratic parameters. Control flags for other protocol-idiosyncratic parameters must be defined for each off-loaded protocol.

Responses

The following responses have been identified for the Status command:

000	Command completed successfully
201	Command not appropriate
300	Problem with multiple chunks
301	Syntax problem with Command
302	Inappropriate status request
303	Inappropriate status response
401	Protocol Interpreter in OPE not responding
402	Failure in remote protocol interpreter
403	Failed; insufficient protocol interpreter resources
501	Failed; insufficient OPE resources
601	Request violates security policy
9xx	Protocol Idiosyncratic status responses

Example of Status Command

The status command can be particularly complex, depending on the protocol and particular type of status information. One possible use of the status command when off-loading TCP is to communicate the status service request. For performing this operation the status command would be:

C ST Q <nl>

Notes to the Implementor

End

Purpose of the End Command

The purpose of the End command is to communicate that services of the off-loaded protocol are not required. The semantics of the End command depends upon whether it was issued by the Host or the OPE.

- If the Host issues the End command, it means the process in the Host no longer requires the services of the offloaded protocol.

- If the OPE issues the End command, it means the remote entity has no more data to send (e.g., the off-loaded protocol is TCP and the remote user has issued a TCP close).

Parameters of the End Command

One parameter is associated with the End Command. It indicates whether the termination should be "graceful" or "abrupt" (see below).

Graceful/Abrupt

The Graceful/Abrupt parameter indicates whether the End should be handled gracefully or abruptly. If it is handled gracefully, then data in transit is allowed to reach its destination before service is actually terminated. An abrupt End occurs immediately; all data transmitted from the Host but still pending in the OPE is discarded, and no new incoming data is sent to the Host from the OPE.

The parameter is indicated by a single ASCII character. The character "G" denotes graceful, and "A" denotes abrupt. The default value for this parameter is graceful.

Responses

The following responses have been identified for the End command:

000	Command completed successfully
201	Command not appropriate
300	Problem with multiple chunks
301	Syntax problem with Command
302	Illegal Type of End Command
401	Protocol Interpreter in OPE not responding
402	Failure in remote protocol interpreter
403	Failed; insufficient protocol interpreter resources
501	Failed; insufficient OPE resources
601	Request violates security policy

Additionally, protocol idiosyncratic responses will be defined for each off-loaded protocol.

Example of End Command

The syntax of the End command is relatively straightforward. It consists of a chunk that contains only a chunk usage identifier, the end command string, and the parameter indicating whether the end should be graceful or abrupt. A possible valid (abrupt) End command would be:

C EN A <nl>

Notes to the Implementor

Once an End has been issued in a given direction any other commands on the channel in the same direction are in error and should be responded to appropriately.

No-op

Purpose of the No-op Command

The No-op command performs no operation. Its purpose is to permit the Host and OPE to participate in a dialog which does not alter the state of communication activities, both for debugging purposes and to support features of certain protocols (e.g., Telnet's Are You There command).

Parameters of the No-op Command

There are no parameters associated with the No-op command.

Responses

There are only two possible legal responses to the No-op command. They are:

```
000    No-op Command Completed Correctly
300    Problem with multiple chunks
```

Example of No-op Command

Syntactically the No-op command is quite simple. It consists of a chunk that contains only the chunk usage identifier and the string for the command, and the newline. One possible valid No-op command is:

```
C NO <nl>
```

Notes to the Implementor

No-ops are included for use in testing and initial synchronization. (The latter use is not mandatory, however. That is, no exchange of No-ops is required at start-up time, but it is conceivable that some implementations might want to do it just for exercise.) They are also traditional.

References

(References [1]-[3] will be available in M. A. Padlipsky's "The Elements of Networking Style", Prentice Hall, 1985.)

[1] Padlipsky, M. A., "The Host-Front End Protocol Approach", MTR-3996, Vol. III, MITRE Corp., 1980.

[2] Padlipsky, M. A., "The Elements of Networking Style", M81-41, MITRE Corp., 1981.

[3] Padlipsky, M. A., "A Perspective on the ARPANET Reference Model", M82-47, MITRE Corp., 1982.

[4] Bailey, G., "Network Access Protocol", S-216,718, National Security Agency Central Security Service, 1982.

[5] Day, J. D., G. R. Grossman, and R. H. Howe, "WWMCCS Host to Front End Protocol", 78012.C-INFE.14, Digital Technology Incorporated, 1979.

APPENDIX

Per-Protocol Offloading Descriptions

1. Command Level Interface to an Off-loaded TCP

This appendix discusses the use of the commands described in the body of this document to provide an interface between a Host process and an off-loaded interpreter of the DoD's Transmission Control Protocol (TCP). The interface described here is functionally equivalent to the interface found in the MIL-STD 1778 specification of TCP. It is not, however, identical, in that some features of the interface are particularly relevant only in an inboard implementation.

The first section describes the mapping between the interface events of MIL-STD 1778 and the commands and responses of this H-FP, and highlights the unique features of the interface. The next sections discuss the details of each command. These details include the specialized usages of the command and the protocol-idiosyncratic parameters for that command.

1.1. Relation to MIL-STD 1778 Interface

Most of the requests and responses of the TCP interface specified in MIL-STD 1778 are mapped directly to H-FP Commands and responses. The exceptions are noted in the following descriptions.

1.1.1. Requests

Unspecified Passive Open, Fully Specified Passive Open, Active Open, and Active Open with Data requests are all implemented using variations of the Begin command. The distinction between Passive and Active Open is made using the Active/Passive parameter of Begin. The distinction between unspecified and fully specified lies in the presence or absence of the destination address fields. An active open with data is identical to a normal active open, except for the presence of data following the command.

The Send Service Request is implemented using the Transmit command. Special protocol idiosyncratic parameters are provided for Urgent, Push, and changing the ULP timeout action and values. The response to the Transmit command indicates that the appropriate Send call has been made.

There is no corresponding response in the specified TCP interface; its only significance is that the Host can issue another Transmit command.

The Allocate event is a specification feature of MIL-STD 1778 to indicate the willingness of the user to accept incoming data across the interface. However, because this is precisely the type of flow control provided by the Channel level, the Allocate event would be a superfluous mechanism. Thus, there is no direct analogy to that event in the H-FP interface. A Host process indicates its willingness to accept new data by informing the channel via its flow control interface (if it has an explicit one).

Close and Abort are provided by the End command. Close uses the graceful version of the End command, while Abort uses the abrupt version. The response indicates that the End command has been received and the corresponding Close or Abort was issued. There is no corresponding response in the specified TCP interface.

Status is provided by using the query form of the Status command. The response to the Status command contains the information (see below).

1.1.2. Responses

The Open Id response is provided so that the user has a shorthand name by which to refer to the connection. With an outboarded TCP interpreter, there is a one-to-one mapping between TCP connections and H-FP channels. Hence, the Open Id event is not needed, since the channel ID is sufficient to indicate the desired connection.

The Open Failure and Open Success responses are provided using OPE-generated responses to Begin commands (which provide the Active and Passive Service response primitives) issued by the Host. The value of the response code indicates whether the Begin command succeeded or failed, and can be mapped to the appropriate Open Failure or Open Success indication by the Host.

Deliver is provided by having the OPE issue a Transmit command. As mentioned above, the "flow control" between the TCP interpreter and the Host is provided by the Channel layer, so no explicit interface events are needed. The

response to the Transmit command indicates the data was received by the Host process. There is no corresponding response in the specified TCP interface.

The Closing and Terminate service responses are provided using the End command. Closing is indicated using the graceful version of the command, while terminate is provided using the abrupt version. The response indicates the End command was received by the Host process. There is no corresponding response in the specified TCP interface.

Status Response is provided by a response to the query version of the Status command. The status information is communicated via protocol-idiosyncratic parameters following the Response code.

Error messages are reported using the spontaneously generated version of the Status command issued by the OPE. The error message is provided in a parameter. The response indicates the error message was received by the Host process. There is no corresponding event in the specified TCP interface.

1.2. The Begin Command

The Begin command is used in TCP in three major ways:

1. To inform the OPE that a process in the Host wishes to open a connection to a particular port on a internet address.

2. To inform the OPE that a process in the Host wishes to be informed when a connection attempt is made to any or to a specific port at this Host's internet address.

3. To inform the Host that a connection attempt to the OPE has arrived, and there was no Begin of the second type (passive open) issued by the Host relevant to that particular port.

1.2.1. Specialized Usage

There are four major aspects to the specialized usage of the Begin command and its parameters. These parameters are:

1. The meaning of the Mediation Level parameter

2. The selection of blocking treatment of Transmit command

3. The meaning of the address components

4. The selection of the TCP Active Open with Data primitive.

The Mediation Level parameter has only two possible values when offloading TCP. These are "9" and "0". The normal usage of an off-loaded TCP uses the value "9", which means the Host is in no way involved in the operation of TCP. The value "0" indicates the Host wishes to negotiate with the TCP options.

The normal TCP Send event is non-blocking. That is, when a user issues the send command, it counts on the reliability services of TCP, and is not explicitly notified when the data has reached the other end of the connection and been properly acknowledged. Hence, the default value for this parameter with TCP is "N". There are some applications where the user may not wish to receive a response to a Transmit command until the data has been acknowledged by the other end of the connection. In these cases, the value "B" should be used for this parameter. If such a feature is not supported by the offloaded TCP interpreter, then it is acceptable to issue a 100 level Conditional acceptance indicating that blocking is not supported, but the Begin command will proceed using non-blocking Transmits.

The primary address components of the local and remote addresses refer to the internet addresses of (or a symbolic Host name for) the respective Hosts. The secondary components refer to the particular sockets at those internet addresses. Normally, the secondary components (ports) are specified numerically. They may, however, be specified by name if the port is a well-known service port. In an Active Begin command, the remote addresses primary and secondary components must be specified. The local address components need not be specified, unless the user wishes to indicate that the connection should be from a particular port or a particular internet address of a multi-homed Host. In a Passive Begin command, the remote addresses are specified only if connection attempts from one particular Host are of interest. The local address secondary component must be used to indicate on which port to perform the Listen.

The way the TCP Active Open with data is provided is by including the data with the Begin Command. This data is included in the same Channel level chunk, immediately following the newline. If the data is more than a single chunk can hold, then the multi-chunk command feature of the H-FP must be used.

1.2.2. Protocol-Idiosyncratic Parameters

The protocol-idiosyncratic parameter identified for the TCP interface is the "ULP timeout" information. This information includes whether the offloaded interpreter should abort the connection on a ULP timeout or report it to the inboard user, and also the numerical value of the timeout interval. The format chosen for this parameter is a single letter followed immediately (with no spaces) by an ASCII number. The letter can be either "R" or "A", and indicates that the ULP timeout should cause a report or an abort, respectively. The number is interpreted to be the timeout interval in seconds.

1.2.3. Examples of the Command

An example of an Active Begin command that might be issued by an inboard user Telnet is:

```
C BE TCP A ISIA 9 N 23 ,, 60 R 0 -pi R120 <nl>
```

ISIA is the destination Host, 23 is the well-known port number for Telnet connections, a Begin timeout of 60 seconds was chosen. The desired type of service is to strive for good response time, the transmissions are expected to be in small units, and protocol-idiosyncratic parameter R120 implies that a ULP timeout of 120 seconds should be reported.

An example of a Passive Begin Command that might be issued by an inboard server Telnet is:

```
C BE TCP P ,, 9 N ,, 23 ,, R 0 -pi R120 <nl>
```

The major differences are that no remote address components are specified, and the local secondary address component is identified as the socket on which the Listen is being performed. Also, the default ("infinite") timeout is taken.

1.3. The Transmit Command

The Transmit command is used by the Host process to instruct the off-loaded TCP interpreter to send data to a remote site via the TCP connection associated with the command's channel. It is used by the OPE to deliver incoming data from the connection to the process in the Host.

1.3.1. Specialized Usage

The Transmit command must be capable of providing all the specialized features of the Send and Deliver Event. These special features are Urgent, Push, and modification of the ULP Timeout action and/or interval.

Urgent is a means to communicate that some point upcoming in the data stream has been marked as URGENT by the sender. While the actual Urgent bit travels through the connection out-of-band, it carries a pointer that is related to the sequence numbers of the in-band communication. Hence, the urgency must be indicated in the Transmit command rather than the Signal command.

Push is a feature of the TCP Send Event that is used to indicate that the data in the Transmit command should be sent immediately (within the flow control constraints), rather than waiting for additional send commands or a timeout. Push is indicated in the Transmit Command. The push feature has the same meaning when sent from the OPE to the Host. If the Host implementation does no internal queuing, the flag has no meaning.

The TCP Send event permits the user to modify the "ULP timeout action" and/or the "ULP timeout interval" associated with that connection. When changed, the new values take effect for the remainder of the connection, unless changed later with another Send. This feature is provided in this H-FP using the Transmit Command.

1.3.2. Protocol-Idiosyncratic Parameters

The three features identified above are provided using protocol-idiosyncratic parameters.

The first such parameter is the Urgent parameter. From the point of view of the interface, it is just a flag that indicates the data is urgent (the actual Urgent pointer is a

concern of the off-loaded TCP interpreter, which is keeping track of the sequence numbers). When issued by the Host process, the Urgent flag means the stream should be marked. When issued by the OPE, it means the receiver should go to (or remain in) the Urgent receive mode. If the flag is not set in the Transmit issued by the OPE, then the receiver should remain in (or return to) the non-urgent receive mode. The value of this protocol-idiosyncratic parameter is "U" if the Urgent is set, or "N" if it is not set. The default value for this parameter is "N". Since this parameter is the first protocol-idiosyncratic parameter for the Transmit command, it requires no special flag, and can be indicated using the flag -pi.

The second protocol-idiosyncratic parameter is the Push flag. This parameter is only issued by the Host, since there is no Push in the TCP Deliver event. Its value is "P" for push, or "N" for normal. The default value of this parameter is "N". Its control flag is -pu.

The third protocol-idiosyncratic parameter is the ULP timeout action and value parameter. The action part indicates whether the offloaded interpreter should abort the connection on a timeout or report it to the inboard user. The value part is the numerical value of the timeout interval. The format used for this parameter is the same as in the Begin command, which is a single letter followed immediately (with no spaces) by an ASCII number. The letter can be either "R" or "A", and indicates that the ULP timeout should cause a report or an abort, respectively. The number is interpreted to be the timeout interval in seconds. The default interpretation for this parameter is its previous value. The control flag for this parameter is -ul.

1.3.3. Examples of the Command

An example of a Transmit command issued by a Host process is

```
C TR -pi N P R160 <n1> <DATA>
```

where <DATA> is the data contained within the chunk. This command is for a non-urgent but pushed TCP Send event, that also resets the timeout action and interval to Report with a value of 160 seconds. The response mode (i.e., nonblocking) is derived from the Begin command and not effected by transmit.

An example of a Transmit command issued by the OPE is

C TR -pi N <nl> <DATA>

where <DATA> is the data contained within the chunk. This command is for a non-urgent delivery (presumably, after a previous Urgent delivery).

1.4. The Condition Command

The Condition command is used to modify the transmission characteristics of the connection. The parameters that make sense to modify with TCP are the Transmit Response discipline, the Type of Service, and the Flow Control Advice.

1.4.1. Specialized Usage

There is no usage of the Condition command with an offloaded TCP interpreter that is particularly specialized.

1.4.2. Protocol-Idiosyncratic Parameters

There are no protocol-idiosyncratic parameters for the condition command for the off-loaded TCP. It would be possible for the ULP timeout action values to be changed with a condition command. However, this is accomplished with the Transmit command, which more closely models the interface specified in MIL-STD 1778. We propose that the condition command not provide this capability.

1.4.3. Examples of the Command

An example of the Condition command to change the flow control advice for a connection is

C CO -fc 1 <nl>

which indicates that relatively small transmission units are now expected.

1.5. The Signal Command

As we currently understand it, TCP's URGENT feature provides an INband signal rather than a true out-of-band signal (and at least one of us deeply regrets this). The actual URGENT bit is sent out-of-band, but it contains an URGENT pointer which relates the URGENT to its position in the data stream. The actual semantics of the URGENT is left to the higher level protocol (e.g., Telnet says to discard all data up to the URGENT pointer). Since the Signal command is allowed to cross a pending Transmit in the H-FP channel, it would be potentially dangerous to implement the interface to TCP URGENT using the Signal command since the wrong sequence number could be used as the urgent pointer. Barring persuasive arguments to the contrary, it is proposed that Signal should not be used with TCP.

1.6. The Status Command

The Status command maps directly into the TCP Status event when issued by a Host process. It is also used for the TCP error event when issued by the OPE. There is currently some question as to how information from lower protocol levels (e.g., ICMP error messages) should be reported to TCP users. When these issues are resolved, there may be other uses for the Status command. We solicit other ideas for the Status command with this report.

1.6.1. Specialized Usage

The major specialized usage of the Status command is to provide the error reporting service. This usage is a form of the Status generated by the OPE.

1.6.2. Protocol-Idiosyncratic Parameters

When used as a TCP Status request (command issued by the Host process), there are no protocol-idiosyncratic parameters associated with the Status command. The OPE response codes the TCP status.

When used as a TCP error report (command issued by the OPE), there is one protocol-idiosyncratic parameter associated with the Status command. It is an error description in the form of a text string. It requires no special control flag since the flag -pi is unambiguous and there are no other protocol-idiosyncratic parameters.

1.6.3. Examples of the Command

An example of the Status command issued by the Host process to request status information is

C ST Q <nl>

The status information is returned in the response to the status command.

An example of the Status command issued by the OPE to report an error from the TCP interpreter is

C ST R -pi "Connection already exists" <nl>

which is issued when a TCP open (HFP Begin) is issued to an already opened (foreign) connection.

1.7. The End Command

The End command is used to indicate that TCP services are no longer required. Thus, it can be mapped into either the TCP Graceful Close or the TCP Abort events. It is also used as the TCP Closing response (as contrasted with the response by the OPE to the close command), when issued by the OPE.

1.7.1. Specialized Usage

Because of the nature of the two-way close provided by TCP, there is a possibility that the Host and the OPE wish to gracefully terminate the connection at the same instant. If this happens, then both the Host and the OPE would issue End commands at the same time. To be prepared for this, it is necessary to make this the normal graceful closing sequence. In other words, both the Graceful Close request and the Closing response are mapped to End commands, and the response to one of those commands only indicates that the command has been received and executed, but not that the connection is actually fully closed. The connection is gracefully closed when both End commands have been issued, and both successful responses have been received.

With an abrupt end, a two-way exchange is not necessary. Only the Host or the OPE need issue it, for the connection to be aborted.

1.7.2. Protocol-Idiosyncratic Parameters

There are no protocol-idiosyncratic parameters for the End command used with TCP.

1.7.3. Examples of the Command

An example of the End command used to indicate either a TCP Close request (from the Host process) or TCP Closing response (from the OPE) is

```
C EN G <nl>
```

An example of the End command used as an Abort request (from the Host process) or as a Terminate response is

```
C EN A <nl>
```

2. Command Level Interface to an Off-loaded Telnet

This appendix is provided to discuss the use of the commands described in the body of this document to provide an interface between a Host process and an off-loaded interpreter of the Telnet protocol.

The interface described here is not based on a formal interface. There are several reasons for this, including the lack of a widely accepted standard interface to Telnet, and its headerless nature. Consequently, the interface described here is very similar to the actual Telnet data stream.

2.1. The Begin Command

The Begin command is used with Telnet to initiate Telnet connections.

2.1.1. Specialized Usage

There are three major specialized usages to the Begin command. They are the meaning of the Mediation Level parameter, the way the number of incoming Telnet connections are supported, and the meaning of the secondary address components.

The mediation level is used in Telnet to control which of the various Telnet activities are performed by the OPE, and which are controlled by the Host. It has been determined

that all monitoring of the Telnet Socket should be performed by the OPE. Mediation level 9, which is the default, indicates the Host desires to play no role in Telnet operation. Level 5 means that protocol-idiosyncratic parameters to this Begin command indicate which incoming options the Host wishes to handle; all other options, and all NVT translations, are to be performed by the OPE. Level 0 indicates that the Host will handle all options, while all NVT translations are to be performed in the OPE (see Section B.1.3).

The Host can either accept the connections by fielding OPE generated Begins, or by issuing passive Begins to the OPE. The Host may wish to restrict the number of incoming Telnet connections that it will handle at any particular time. It can do this by rejecting OPE-generated Begins above a certain number, or by limiting the number of Host-issued passive Begins. However, precedence constraints dictate that the Host actually issue additional passive Begins or accept additional Begins from the OPE beyond the maximum number it is normally willing to support, so that high-priority service requests can be accommodated, possibly by preempting lower priority activities.

The secondary address component is used to refer to specific ports. Normally, they are used only when the standard or default ports are not used, such as special purpose applications or testing.

2.1.2. Protocol-Idiosyncratic Parameters

The protocol-idiosyncratic parameters to the Telnet Begin command are the identifiers for the options which the host wishes to negotiate when using mediation level 5. On other mediation levels, these parameters are not used.

2.1.3. Examples of the Command

An example of a passive Begin for an outboard Telnet protocol is:

```
C BE TEL P ,, 5 N -fc 0 -pi 9 <nl>
```

Where the parameters are:

TEL	Code for the Telnet Protocol
P	Passive Begin

```
,,      Skip the Foreign Address Primary Component
5       Mediation Level is 5
N       Non Blocking Transmits
-fc     Skips over parameters up to Flow Control Advice
S       Small Blocks are appropriate for Telnet
-pi     Skips over parameters to the Protocol Idiosyncratic
        List of Options to be Handled by the Host.
9       Option Code for Line Length Option
```

Here, no remote address component was specified, since the Host will accept connections from any Host. Similarly, no local addresses are specified, since the default well-known socket for this Host is to be used. In this example, the Host specifies it will handle the line length option (number 9). Other options are handled in the OPE.

An example of an active Begin for an outboard Telnet protocol is:

```
C BE TEL A ISIA 5 N -fc 0 -pi 9 <nl>
```

This command is identical to the passive command, except that a remote primary address component is specified to identify the intended Host. No remote secondary component is specified, since the well-known socket at that Host is to be used. No local secondary address components are specified, since the connection can originate from any available socket of the appropriate type selected by the OPE.

2.2. The Transmit Command

The Transmit Command is used to send data across a Telnet connection.

2.2.1. Specialized Usage

The Transmit command is used to transmit data over the Telnet connection. There is one specialized aspect of the Transmit command used with an outboard Telnet interpreter. This is the provision of the Go Ahead feature of Telnet that supports half-duplex devices.

Go Ahead is provided as a protocol idiosyncratic parameter to the Transmit. It is only used if the Host will support it, however. It is our opinion that Go Ahead is probably not a proper thing for the default case.

Go Aheads are a matter between the Host and the terminal. It is difficult to offload the generation of Go Aheads to the OPE, since the OPE is not really cognizant of the semantics of the communication between the Host and the terminal. Hence, the OPE does not know when the Host is done transmitting and willing to pass "the turn" back to the terminal. Similarly when the remote site relinquishes control, the OPE includes Go Ahead in its TR.

We don't believe this Go Ahead problem to be an indictment against outboard processing. It merely illustrates that functionality not found in a Host cannot necessarily be provided by the OPE. Hence, we provide this note to the implementor: if the Host cannot generate the protocol-idiosyncratic Go Ahead parameter, then the DO Suppress Go Ahead must be issued immediately after the connection is established.

2.2.2. Protocol Idiosyncratic Parameters

The protocol idiosyncratic parameter is the Go Ahead indicator. When present, the character "G" is used to mean the Go Ahead can be sent to the other end of the connection, but only after the data associated with that Transmit command is sent. When the character is any other value, or is absent, the Go Ahead should not be sent.

2.2.3. Examples of the Command

An example of the Transmit command is:

```
C TR -pi G <nl> <DATA>
```

With this command, the Go Ahead is passed to the other side after the data is sent.

2.3. The Condition Command

The Condition command is used with Telnet to modify the Transmission characteristics and to enable or disable Telnet options on a Telnet connection.

2.3.1. Specialized Usage

The Condition command takes on specialized usage with Telnet, in addition to its normal usage. It is used to

control the option selection and negotiation process, when such selection is performed by the Host (currently, this is done at mediation levels 5 and 1, but not at level 9).

A set of protocol-idiosyncratic parameters has been defined for this purpose. They are based heavily on the Telnet negotiation and subnegotiation mechanisms. For simple negotiations there are two parameters, a negotiation type (from the set {DO, DONT, WILL, WONT}) followed by the code (numeric) or name (symbolic) for the desired option. The codes for the options are identified below. A basic difference between the H-FP interface to Telnet and the internal Telnet protocol is that additional parameters are included with the request (DO or WILL). The Telnet protocol subnegotiation is used internally to communicate that information in the Telnet data stream. Option-specific, protocol-idiosyncratic parameters are used for these additional parameters.

Both the Host and the OPE can issue these Condition commands. When issued by the Host, it means the user wishes to enable or disable a particular option. The OPE proceeds to issue the appropriate negotiation commands (i.e., IAC <DO> <code>) in the Telnet data stream. When the results of the option negotiation are available, a response is generated by the OPE. For the types DO and WILL, a 000 Response indicates the appropriate acceptance (WILL or DO, respectively). A nonzero Response code may indicate negotiation failure or negotiation rejection (among other things). For the types DONT and WONT, a 000 Response indicates the option will be disabled. A negotiation rejection should not be expected in those cases.

When the Condition command is issued by the OPE, it means the other end of the connection is negotiating a change. Here the response from the Host indicates the Host's desired action for the option negotiation. Again, valid requests to disable options (DONT and WONT requests) should always get a 000 Response.

2.3.2. Protocol-Idiosyncratic Parameters

There are two protocol-idiosyncratic parameters for primary negotiation using the Condition command. These are the negotiation type and the option code. The negotiation type is one of the set of {DO, DONT, WILL, WONT}. The option code is a numeric value used to identify the particular

option being negotiated. The values for these codes are indicated here, but are identical to the codes used in the actual Telnet negotiation. The codes are:

Option Name	Option Code	Short Name
Transmit Binary	0	Binary
Echo	1	Echo
Suppress Go-Ahead	3	SuppressGA
Approximate Message Size	4	NAMS
Status	5	Status
Timing Mark	6	TimingMark
RCTE	7	RCTE
Line Length	8	LineLength
Page Size	9	PageSize
Carriage Return Disp	10	CRDisp
Horizontal Tabstops	11	HTabStops
Horizontal Tab Disp	12	HTabDisp
Formfeed Disposition	13	FFDisp
Vertical Tabstops	14	VTabStops
Vertical Tab Disposition	15	VTabDisp
Linefeed Disposition	16	LFDisp
Extended ASCII	17	ExASCII
Logout	18	Logout
Data Entry Terminal	20	DET
Terminal Type	24	TermType
Extended options list	255	ExOptions

Options not listed here may of course be used. The code number should be the same as the option code used in Telnet negotiation.

2.3.2.1. Simple Options

Options that do not require additional parameters use the simple negotiation mechanisms described briefly above and in greater detail in the Telnet documentation. No additional parameters are required. These options include the Transmit Binary, Echo, Suppress Go Ahead, Status, Timing Mark, and Logout options.

2.3.2.2. Approximate Message Size Option

The Approximate Message Size option requires two parameters. The first indicates whether the approximate message size being negotiated applies to the local or the remote end of the connection. DS means the size applies

to the sender of the command (i.e., if the Host issues the command, DS means the local end of the connection; if issued by the OPE, DS means the remote end of the connection). DR means the size applies to the receiver of the command (i.e., if the Host issues the command, DR means the remote end; if issued by the OPE, DR means the local end of the connection). This convention is consistent with the Telnet subnegotiation mechanisms. The second character is an ASCII encoded numeric value, which is a character count of the message size.

2.3.3. Line Width and Page Size Options

The Line Width and Page Size Options require two additional parameters. The first indicates whether the line width or page size being negotiated applies to the local or the remote end of the connection, and uses the DS and DR convention described above. The second parameter is an ASCII encoded numeric value, which is interpreted as follows (assuming the Condition command was issued by the Host):

0	The Host requests that it handle length or size considerations for the direction indicated by the first parameter.
1 to 253	The Host requests that the remote end handle the size or length considerations for the direction indicated by the first parameter, but suggests that the value indicated be used as the size or length.
254	The Host requests that the remote end handle the size or length considerations for the direction indicated by the first parameter, but suggests that the size or length be considered to be infinity.
255	The Host requests that the remote end handle the tabstop considerations, and suggests nothing about what the value should be.

If the Condition command is issued by the OPE, then the roles of the Host and the remote end are reversed.

2.3.4. Tabstop Options

The Horizontal and Vertical Tabstops options require two option specific parameters. The first is either DR or DS, as was described previously. The second is a list of one or more ASCII encoded numeric values separated by spaces which, assuming the Condition command is issued by the Host, are individually interpreted as:

0	The Host requests that it handle tabstops for the direction indicated by the first parameter.
1 to 250	The Host requests that the remote end handle the tabstop considerations for the direction indicated by the first parameter, but suggests that the value(s) indicated should be used as the tabstops.
255	The Host requests that the remote end handle the tabstop considerations for the direction indicated by the first parameter, and suggests nothing about what the value should be.

If the Condition command is issued by the OPE, then the roles of the Host and the remote end are reversed.

2.3.5. Character Disposition Options

The Carriage Return Disposition option, the Horizontal Tab Disposition option, the Formfeed Disposition option, the Vertical Tab Disposition option, and the Linefeed Disposition option are all considered character disposition options from the perspective of H-FP. Two option-specific parameters are required for the character disposition options. The first is the DR or DS code, which was described previously. The second is a single ASCII encoded numeric value, which is interpreted as (assuming that the Host issued the Condition command):

0	The Host requests that it handle the character disposition for this connection.
1 to 250	The Host suggests that the remote end handle the character disposition considerations, but suggests that the value indicated should be taken as the number of nulls which should be

inserted in the data stream following the particular format character being subnegotiated.

251 The Host suggests that the remote end handle the character disposition considerations, but recommends that it replace the character with some simplified character similar to but not identical with it (e.g., replace a tab with a space, or a formfeed with a newline).

252 The Host suggests that the remote end handle the character disposition considerations, but recommends that it discard the character.

253 The Host suggests that the remote end handle the character disposition, but recommends that the effect of the character be simulated using other characters such as spaces or linefeeds.

254 The Host suggests that the remote end handle the character disposition considerations, but recommends that it wait for additional data before sending more data.

255 The Host suggests that the remote end handle the tabstop considerations, and suggests nothing about what the value should be.

Some of the codes between 251 and 254 are not used with some character disposition options. Refer to the ARPANET documentation for additional details.

If the Condition command is issued by the OPE, then the roles of the Host and the remote end are reversed.

2.3.5.1. RCTE Option

The Remote Controlled Transmission and Echoing option requires parameters to indicate the sets of break characters and transmit characters. There are two option-idiosyncratic parameters for RCTE. The first is a list of the character classes that make up the set of break characters, as defined in the RCTE documentation. The second is a list of character classes that make up the set of transmit characters, as defined in the RCTE documentation. Since the two classes are optional and

can be of arbitrary length, it is necessary to precede each list with a -bc (break characters) or -tc (transmit characters). The character classes are defined as

```
1 Upper Case Letters   A through Z
2 Lower Case Letters   a through z
3 Digits  0 through 9
4 Format effectors  <BS> <CR> <LF> <FF> <HT> <VT>
5 Non-format control codes, plus <ESC> and <DEL>
6 Punctuation  . , ; : ? !
7 Grouping    { [ ( < > ) ] }
8 Misc  '  " / \ % @ $ &  + - * = ↑ ← | ~
9 <space>
```

2.3.5.2. Extended Option List

The Extended Option List option requires a parameter to carry the number of the option on the extended list. There is thus one option specific parameter to the Condition command when used for this purpose, which is the number of the option on the extended option list. It can be expressed in ASCII using an octal, decimal, or hexadecimal format.

2.3.5.3. Terminal Extension Options

The Extended ASCII, SUPDUP, and Data Entry Terminal options of Telnet were all attempts to extend the basic capabilities of the Telnet data stream beyond the simple, scroll mode terminal model that was the basis of the original Telnet design.

All of these options have limitations to their effectiveness. The Extended ASCII option lacks a standardized interpretation of the bit patterns into extended ASCII characters. The SUPDUP effort was actually an independent mode where a different virtual terminal protocol was used, and the option was there merely to switch to and from this protocol. The Data Entry Terminal option requires the excessive overhead of subnegotiation for each use of extended features. All of these options lack the more valuable asset of widespread implementation and use.

The way these options should be handled is not detailed in this appendix. It is clear that the Condition command could be used for initiating and terminating the use of

these options. The actual transmission of characters related to the extended terminal features should be provided by the Transmit command, either as part of the normal Host-to-OPE data stream or by using protocol-idiosyncratic parameters.

A more recent option, the Terminal Type option, should be mentioned here. It permits one end of a connection to request information about the terminal at the other end or send information about the terminal at the local end. This is convenient for systems that provide a wide variety of terminal support, but it clearly does not follow the model of reducing the MxN problem by use of a virtual terminal. Its use is very straightforward in the H-FP context. It only requires sending the terminal type to the other end, and activating the Binary Transmission Option.

2.3.5.4. Status Option

The Status option is enabled using the negotiation mechanism of Telnet. However, the means to transfer status information between OPE and the Host is provided via the Status command. Therefore, details of status negotiation are irrelevant to the interface to the outboard Telnet.

2.3.6. Examples of the Command

The following example shows the command issued by a Host to the OPE, requesting that the OPE negotiate with the other side so that remote echo is performed.

```
C CO -pi DO 1 <nl>
```

The numeral 1 is the option code for ECHO from the table above. All of the simple options listed above use this same basic format.

The options with additional parameters use straightforward extensions of this syntax. For example, a possible usage of Condition by the Host to set the approximate message size is:

```
C CO -pi DO 4 DS 1024
```

The 4 is the Option Code for the Approximate Message Size option, the DS indicates that Host's message size should be set, and 1024 is the desired size.

2.4. The Signal Command

The Signal command is used with Telnet to provide the Telnet Interrupt Process and Abort Output services.

2.4.1. Specialized Usage

The Signal command is used with an outboard Telnet interpreter to interface to the Telnet synch mechanism. This mechanism is used with a protocol-idiosyncratic parameter, which indicates what particular command is being "synched." It is expected that normally, this Signal mechanism will only be used with the Interrupt Process and Abort Output Telnet signals. When the Signal command is issued by the Host, it goes through the Channel (out-of-band) to the OPE, where the Telnet interpreter issues the corresponding Telnet signal and synch sequence. When such a sequence is received by the OPE, it immediately issues a Signal to the Host. It is expected that a Host or OPE would not, in general, reject the Signal command unless it is badly formed.

2.4.2. Protocol-Idiosyncratic Parameters

The Telnet protocol-idiosyncratic parameter used with the Signal command identifies which Telnet signal is begin issued. Normally, it would have the value of either "IP" or "AO", for Interrupt Process or Abort Output. If absent, the default value is "IP".

2.4.3. Examples of the Command

An example of a Telnet Signal Command (in this case, to send an Interrupt Process signal) is:

```
C SI IP <nl>
```

2.5. The Status Command

The Status command is used with Telnet to obtain information about the Telnet connection and the options in effect.

2.5.1. Specialized Usage

The Status command has one specialized aspect when used to interface to an outboard Telnet interpreter. That is to send and receive the Telnet Protocol status request subnegotiation message to and from the data stream. In order to invoke the status command for this purpose, however, the user must have previously issued the Condition Status command, which causes the ability to request status to be negotiated. The OPE, when it receives a valid Status request command, immediately responds to the user indicating the status. The OPE can issue a status to request the Host's negotiated positions.

2.5.2. Protocol-Idiosyncratic Parameters

There are no protocol-idiosyncratic parameters to the Status query command. The Status Response command has a single protocol-idiosyncratic parameter. It is an ASCII string containing the status of the various options (not at their default values).

2.5.3. Examples of the Command

An example of a Status Query command is:

```
C ST Q
```

An example of a Status Response command is:

```
F ST R "WILL ECHO  DO SUPPRESS-GO-AHEAD
L WILL STATUS  DO STATUS" <nl>
```

In the previous example, note the opening quote is in the first chunk, and the closing quote is in the last chunk. This technique permits parameters to span chunk boundaries.

2.6. The End Command

The End command is used to terminate the Telnet connection, either gracefully or abruptly.

2.6.1. Specialized Usage

The graceful termination of a Telnet requires End commands to be issued by both the Host and the OPE. This specialized usage is identical to that of the outboard TCP interface, however.

2.6.2. Examples of the Command

An example of the graceful End command is:

```
C EN G <nl>
```

The abrupt End command is similar.

2.7. The No-op Command

The No-op command is used with Telnet so the Host can determine if the OPE is active, and vice versa.

2.7.1. Specialized Usage

The No-op command has one specialized usage when offloading Telnet. This is to provide the Telnet Are You There (AYT) feature. When an (AYT) message is received by the OPE, it issues a No-op command to the Host. Upon receiving the response from the Host, the appropriate response is sent back in the data stream.

2.7.2. Protocol Idiosyncratic Parameters

There are no protocol-idiosyncratic parameters to the No-op command.

2.7.3. Examples of the Command

An example of the No-op command is:

```
C NO <nl>
```

3. FTP Offloading

   TBS

4. Mail Offloading

   TBS

5. Whatever Offloading

   TBS

Where TBS nominally = To Be Supplied, but really means: We'll argue through these once we get sufficiently positive feedback on the others (and on the H-FP as a whole).